GBNSC:
The GigaBit Network Switch Controller

Presented by:

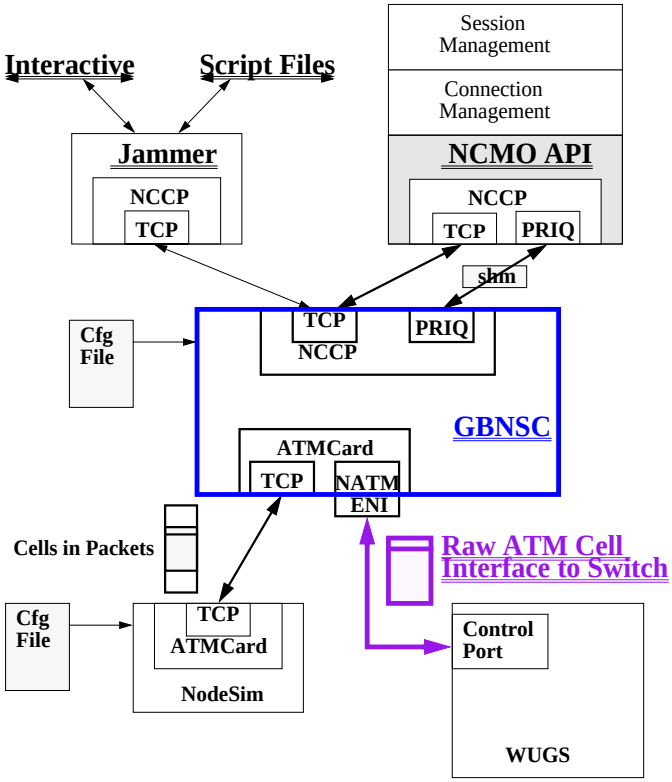
John DeHart
jdd@arl.wustl.edu
http://www.arl.wustl.edu/~jdd
http://www.arl.wustl.edu/arl

Applied Research Laboratory

WUGS Kits Program
Washington University
July 13-24, 1998
August 3-14, 1998

Washington
WASHINGTON • UNIVERSITY • IN • ST • LOUIS

WUGS Software Overview



GBNSC: Purpose

- **Control one WUGS**
- **Hide Hardware details as much as possible from layers above**
- **Provide access to all hardware details to Jammer**
- **Monitor state of switch**
-

GBNSC: Command Line Arguments

options:

-[hH]	**: Get the usage message**
-[rR]	**: Use raw mode. Only send one ping cell through switch at startup**
-[sS] simHostName	**: Hostname where NodeSim is running.**
-p tcpPort	**: TCP port on which NodeSim is listening.**
-P tcpPort	**: TCP port that we should listen on. (Overrides config file!)**
-[dD] debugLevel	**: Controls how much debug output is printed.**
-[nN] ncmoNumber	**: Parent sets NCMO index for synchronization with child GBNSC.**
-[iI] nodeIdentifier	**: Node ID. Helpful in building trees of NCs and GBNSCs.**
-[mM] nodeIdMask	**: Node ID Mask. Helpful in building trees of NCs and GBNSCs.**
-Q inputPriqueID	**: Input (from parent) PRIQUE shared memory ID.**
-q outputPriqueID	**: Output (to parent) PRIQUE shared memory ID.**
filename	**: GBNSC configuration file name.**

GBNSC: Configuration File

- **yacc/lex parsed text file**
- **provides for flexibility as WUGS prototypes evolve**
- **specifies information needed about other end of links for Connection Management to operate**
- **Specifies the control interface for switch**
- **Allows for overrides of some default switch configurations.**
-

Sample Configuration File

```
-- Switch identifier and hardware section

TCPPORT 5550 5561              -- use this port for TCP communications
PORTS 8                -- 8 ports, numbers 0 to 7

CHIPS 1 2              -- Default chips, IPP and OPP
      IPP CHIP 3       -- Overrides for specific ports: chip number,
            6      -- then ports
            7
END


-- Parameters (switch management) section
PARAMS
      -- VPT and switch POLLING are not currently supported
      VPT 1                -- This VPI is VP-terminated
      CONFIGURATION TIMEOUT 60       -- 60 seconds to get it configured
     POLLING TIMEOUT 60              -- 60 seconds between polling switch
END


-- Control connection section
CONTROL
      0 0           -- CP is connected to these two ports
      0/32           -- cntl VXI (cells sent to switch)
      0/32           -- rhdr VXI (cells received from switch)
END
```

Sample Configuration File (con't)

```
- Link information useful for CM applications
LINKS
      0 <=> UNI @  155 @ 1200 "ackbar" 1
      1 =>  UNI @  620 @ 1200 "piett" 2
      2 <=> NNI @ 1200 @  620  0 4 "wooster"
      3 <=  UNI @  620 @  620 "piett" 3
      4 <=> NNI @ 1200 @  620  0 3 "gussie"
      5 <=> NNI @ 2400 @    0  1 3 "jeeves"
      6 <=> NNI @ 2400 @    0  2 6 "r2d2"
      7 <=> NNI @  620 @ 1200  0 2 "wooster"
END



-- Initial configuration section (set the initial values of the config MR fields)
INIT
      IPP 0 VPCOUNT 255          -- Use this value for VPCount in IPP 0
      IPP 1 VPCOUNT 255          -- Use this value for VPCount in IPP 1
      IPP 2 VPCOUNT 255          -- Use this value for VPCount in IPP 2
      IPP 3 VPCOUNT 255          -- Use this value for VPCount in IPP 3
      IPP 4 VPCOUNT 255          -- Use this value for VPCount in IPP 4
      IPP 5 VPCOUNT 255          -- Use this value for VPCount in IPP 5
      IPP 6 VPCOUNT 255          -- Use this value for VPCount in IPP 6
      IPP 7 VPCOUNT 255          -- Use this value for VPCount in IPP 7
END

}
```


Sample Start Up of GBNSC

```
deak[116]> ../NetBSD/GBNSC config.port2

        Defined a  UNI link on port 0
        Defined a  UNI link on port 1
        Defined an NNI link on  port 2
        Defined a  UNI link on port 3
        Defined an NNI link on  port 4
        Defined an NNI link on  port 5
        Defined an NNI link on  port 6
        Defined an NNI link on  port 7
    Switch controller for GBN switch 0.1
    Controller is READY
    Switch is ALIVE
    Switch has 8 Ports
    CP connected to IPP 2  OPP 2
    Control path to the switch via port 2 on VPI/VCI 0/32
    Control path from the switch via port 2 on VPI/VCI 0/32
    GBNSC is listening on TCP Port 3550
```

NCCP Interface from/to Jammer

- **NCCP Ops which map directly to control Cells:**
  - **NCCP_Operation_Ping :**
  - NCCP_Operation_ReadVCXT :
  - NCCP_Operation_ReadVPXT :
  - NCCP_Operation_WriteVCXT :
  - NCCP_Operation_WriteVPXT :
  - NCCP_Operation_ReadMR :
  - NCCP_Operation_WriteMR :
  - **NCCP_Operation_Reset :**
  - NCCP_Operation_TestCell0 :
  - NCCP_Operation_TestCell1 :
  - NCCP_Operation_TestCell2 :
  - NCCP_Operation_ClearMR :
  - NCCP_Operation_ReadVCXTCC :
  - NCCP_Operation_ReadVPXTCC :
  - NCCP_Operation_WriteVCXTCC :
  - NCCP_Operation_WriteVPXTCC :
  - NCCP_Operation_WriteVCXTTR :
  - NCCP_Operation_ClearVCXT :
  - NCCP_Operation_ClearVPXT :
  - NCCP_Operation_TestMR :
  - NCCP_Operation_TestVPXT :
  - NCCP_Operation_TestVCXT :
  - NCCP_Operation_WriteVPXTTR :
  - NCCP_Operation_ReadErrors :
  - NCCP_Operation_ClearErrors :
- **NCCP Ops which map to multiple control Cells:**
  - NCCP_Operation_ClearMRAll :
  - NCCP_Operation_ClearMRAllPPs :
  - NCCP_Operation_ClearVXTAll :
  - NCCP_Operation_ClearVXTAllPPs :
- **Others:**
  - NCCP_Operation_GetCells :
  - NCCP_Operation_PutCells :
  - NCCP_Operation_ShutdownSC :
  - **NCCP_Operation_Ping :**
  - **NCCP_Operation_Reset :**

NCCP Interface with Jammer

- **Interactions with Jammer are strictly Request/Response with Jammer initiating the Request.**
- **When GBNSC receives a Request message it creates a Requestee object to process it.**
- **When the processing is done a Response message is created and sent back to Jammer.**
- **The Requestee objects are “scheduled” so they can time out.**
- **Each Requestee class has the same form:**

NCCP Interface with Jammer (con't)

```
class GBNSC_RequesteeReadVCXT : public GBNSC_RequesteeBase {

    // Additional class data
    SC_PortId    rdPort;// cell reads this IPP (port number
    SC_VCI       rdVCI;// cell reads this VCI

    // Constructor, destructor
    public:
        GBNSC_RequesteeReadVCXT(NCCP_ChannelAddress *theSource, ByteBuffer &buf)
          : GBNSC_RequesteeBase(theSource, GBNSC_CMDATA_NCCP) {
              retrieve(buf);
        }
        virtual ~GBNSC_RequesteeReadVCXT() { }

    // The virtual functions that we must provide
    public:
        virtual void retrieve(ByteBuffer & buf);
        virtual Boolean handle();
        virtual void initiate();
        virtual void receiveCell(GBNSC_ReceiveControlCell & cell);
        virtual int timed_out();
}; // class GBNSC_RequesteeReadVCXT
```


NCCP/NCMO Interface with CM (or NC)

- **NCCP Ops from GBNSC to CM/NC**
  - NCCP_Operation_ChildStatus : SC reports its state (Alive, Dead, ...)
  - NCCP_Operation_LinkStatus : SC reports each link
- **NCCP Ops from CM/NC to GBNSC**
  - NCCP_Operation_ReserveMP : CM reserves a connection
  - NCCP_Operation_UpdateHardwareMP : CM updates a connection into HW
  - NCCP_Operation_RollbackMP : CM undoes a reserve

TCP Interface to NodeSim

- **Interface to NodeSim is encapsulated within ATMCard Object**
- **GBNSC is only controlling one Switch or one NodeSim per execution.**
- **Global state variable defines which it is base on whether the command line options included -[Ss] and -p.**
- **TCP Packet to NodeSim includes:**
  - Cell packed into a ByteBuffer
  - port on which the cell should arrive.
  - VPI/VCI on which the cell should arrive.
- **NodeSim then processes the cell as if it arrived on the Port and VPI/VCI.**

Native ATM Interface to WUGS (Chuck's ENI)

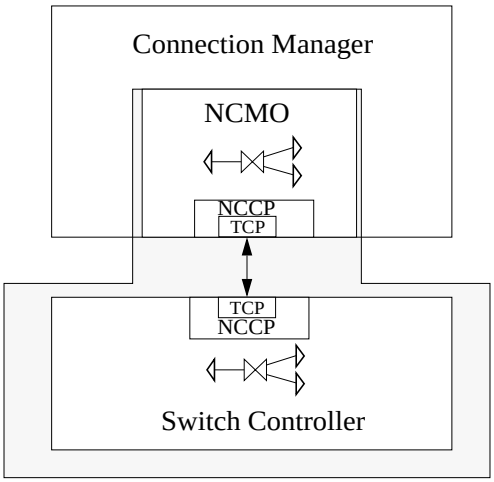
- **Socket interface for sending/receiving RAW cells**
- **Cells can be sent/received with or without header**
- **Cells can be sent/received individually or bunched**
- **For more details on Chuck Cranor's NATM interface to the Efficient Networks Inc. adapter card, please see:**
  http://www.ccrc.wustl.edu/pub/chuck/bsdatm/wucs.html
  http://www.efficient.com/drvdisclaim.html
- **NB: Efficient Networks Inc, takes not responsibility for this third party driver. DO NOT expect them to answer any questions.**

Setting up a NATM Socket (Chuck's ENI)

```
int
setupNATMsocket(char *interface,
                int             aal,
                u_int8_tvpi,
                u_int16_tvci,
          int direction)
{
      struct sockaddr_natm snatm;
      int s_fd;
      int r;
      struct hostent *addressInfo;
      const unsigned char ioctl_ON = 1;

      // Is it r/w or uni-directional?
      s_fd = socket(AF_NATM, SOCK_STREAM, aal);

      if (s_fd < 0)
      {
        perror("socket");
        return(-1);
      }

      bzero(&snatm, sizeof(snatm));
      snatm.snatm_len = sizeof(snatm);
      snatm.snatm_family = AF_NATM;
      sprintf(snatm.snatm_if, interface);
      snatm.snatm_vci = vci;
      snatm.snatm_vpi = vpi;  // Has to be 0 ?

      r = connect(s_fd, (struct sockaddr *) &snatm, sizeof(snatm));

      if (r < 0)
      {
            close(s_fd);
              return(-2);
      }
      ioctl(s_fd, FIONBIO, &ioctl_ON); // set non-blocking IO to ON

      return(s_fd);
}
```


NCMO API

- **Model used for object interface between CM/NC/SC processes.**
- **API in parent process provides access to objects that are created in child.**



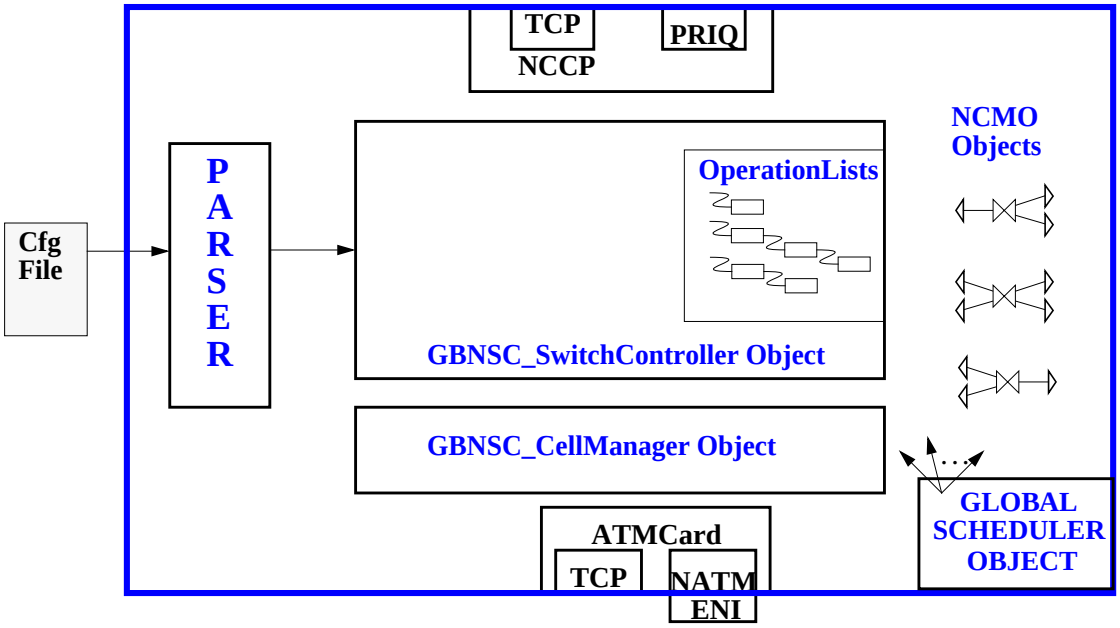
NCMO API

- **Objects provided to parent:**
  - NCMO_Manager : manages multipoint NCMO_Objects
  - NCMO_Object : initiates and communicates with one child process
  - NCMO_Link : represents a switch port
  - NCMO_Multipoint : represents a general multipoint-to-multipoint connection
  - NCMO_Source : represents one transmitter in a connection
  - NCMO_Sink : represents one receiver in a connections
- **Parent process does not know anything about HOW connection is realized in hardware. (i.e. that it uses a binary recycling tree.)**
- **Child process (GBNSC) takes care of allocating any additional resources needed by the particular hardware it is using.**
- **Request/Reserve, Commit/Update protocol using NCCP**
- **Exercise using this API later in the course...**

NCMO API Sample Code

```
source = links[ inLink]->newSource(VC_Connection,  inVpi,  inVci, USE_MINE);
sink = links[outLink]->newSink(VC_Connection,  outVpi,  outVci, USE_MINE);

// connect them to the multipoint object
source->connectToMP(mp);
sink->connectToMP(mp);

// allocate the source and sink resources. this sets the VPI/VCI and valid fields of them.
// the valid field tells the GBNSC whether to allocate the VPI/VCI for us or to use ours
source->alloc( inVpi,  inVci, USE_MINE);
sink->alloc(outVpi, outVci, USE_MINE);

// now reserve the MP Object. This sends the request to the NCMO child (GBNSC).
mp->reserve();

// This sends the request to the NCMO child (GBNSC).
// we will get a rendezvous (call to REQ_rendezvous) when it is done.
// now we wait for the result from NCMO/NCCP

// Some time later.....

// commit the source and sink objects. This marks them as ready to go into hardware
source->commit();  sink->commit();

// now perform the update. This causes the updateHardware request to go down to
// the NCMO child and put the connection into hardware
mp->updateHardware();
```

GBNSC: Internal Code Structure



GBNSC: main() while loop

```
// Main loop
while (GBNSC_SwitchController.getChildStatus() != NCMO_ChildStatus_ChildDied) {
  if (GBNSC_CellManager.check() ||
        (( selectReturnValue = (GBNSC_SwitchController.getSwitcher())->DoSelect()) > 0)) {
        GBNSC_SwitchController.getScheduler()->touch();
        GBNSC_CellManager.touch();
        NCCP_Global.touch();
        if (GSMP_engine != NULL)
          GSMP_engine->touch();

        for (ct = GBNSC_SwitchController.getToBeInitiatedList().getLength(); ct > 0; ct--) {
           if ((op = GBNSC_SwitchController.getToBeInitiatedList().deque()) != (GBNSC_RequesteeBase *)NULL) {
                op->initiate();
           }
        }
        for (ct = GBNSC_SwitchController.getCompletedList().getLength(); ct > 0; ct--) {
           if ((op = GBNSC_SwitchController.getCompletedList().deque()) != (GBNSC_RequesteeBase *)NULL) {
                delete op;
            }
         }
        GBNSC_SwitchController.getScheduler()->touch();
   }  // end of if (... DoSelect())
  else if (selectReturnValue < 0) {
          Sys_Print(ERR_WARNING, "GBNSC", "main", "Select returned error, errno= %d \n", errno);
   }
}  // end of while
```

Miscellaneous